

Nagoya Termination Tool^{*}

Akihisa Yamada¹, Keiichirou Kusakari², and Toshiki Sakabe¹

¹ Graduate School of Information Science, Nagoya University, Japan

² Faculty of Engineering, Gifu University, Japan

Abstract. This paper describes the implementation and techniques of the Nagoya Termination Tool, a termination prover for term rewrite systems. The main features of the tool are: the first implementation of the weighted path order which subsumes most of the existing reduction pairs, and the efficiency due to the strong cooperation with external SMT solvers. We present some new ideas that contribute to the efficiency and power of the tool.

1 Introduction

Proving termination of term rewrite systems (TRSs) has been an active field of research. In this paper, we describe the *Nagoya Termination Tool* (NaTT), a termination prover for TRS, which is available at

<http://www.trs.cm.is.nagoya-u.ac.jp/NaTT/>

NaTT is powerful and fast; its power comes from the novel implementation of the *weighted path order* (WPO) [25, 26] that subsumes most of the existing reduction pairs, and its efficiency comes from the strong cooperation with state-of-the-art *satisfiability modulo theory* (SMT) solvers. In principle, any solver that complies with the SMT-LIB Standard³ version 2.0 can be incorporated as a back-end into NaTT.

In the next section, we recall the dependency pair framework that NaTT is based on, and present existing techniques that are implemented in NaTT. Section 3 describes the implementation of WPO and demonstrates how to obtain other existing techniques as instances of WPO. Some techniques on cooperating with SMT solvers are presented in Section 4. After giving some design details in Section 5, we assess the tool by its results in the *termination competition*⁴ in Section 6. Then we conclude in Section 7.

2 The Dependency Pair Framework

The overall procedure of NaTT is illustrated in Figure 1. NaTT is based on the

^{*} Full version of the paper which is to appear in the *Proceedings of the Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications (RTA-TLCA '14)*, LNCS Advanced Research in Computing and Software Science, Springer, 2014.

³ <http://www.smtlib.org/>

⁴ http://termination-portal.org/wiki/Termination_Competition

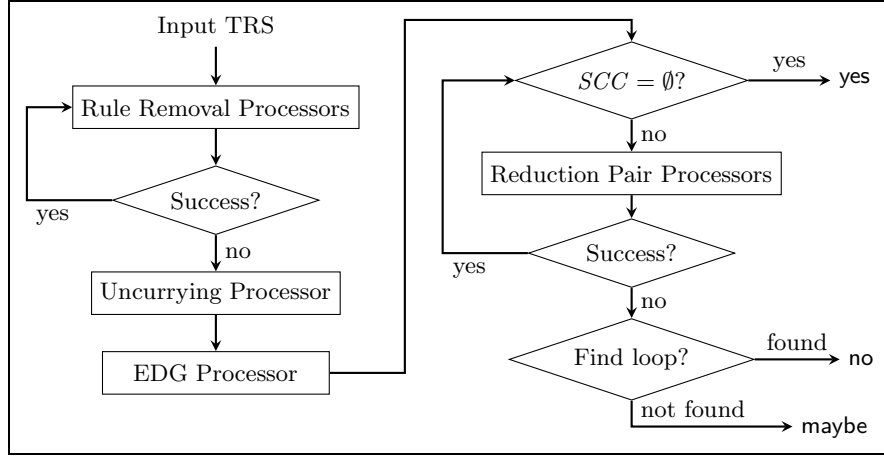


Fig. 1. Flowchart of NaTT

dependency pair framework (DP framework) [1,9,10], a very successful technique for proving termination of TRSs which is implemented in almost all the modern termination provers for TRSs. In the DP framework, dependencies between function calls defined in a TRS \mathcal{R} is expressed by the set $\text{DP}(\mathcal{R})$ of *dependency pairs*. If a function f is defined by a rule

$$f(s_1, \dots, s_n) \rightarrow C[g(t_1, \dots, t_m)] \in \mathcal{R}$$

where g is also defined in \mathcal{R} , then this dependency is described by the following dependency pair:

$$f^\#(s_1, \dots, s_n) \rightarrow g^\#(t_1, \dots, t_m) \in \text{DP}(\mathcal{R})$$

The DP framework (dis)proves termination of \mathcal{R} by simplifying and decomposing *DP problems* $\langle \mathcal{P}, \mathcal{R} \rangle$, where initially $\mathcal{P} = \text{DP}(\mathcal{R})$. To this end, many *DP processors* have been proposed. NaTT implements the following DP processors:

2.1 Dependency Graph Processor

This processor decomposes a DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$ into $\langle \mathcal{P}_1, \mathcal{R} \rangle \dots \langle \mathcal{P}_n, \mathcal{R} \rangle$ where $\mathcal{P}_1, \dots, \mathcal{P}_n$ are the *strongly connected components (SCCs)* of the *dependency graph* [7,10]. Since the dependency graph is not computable in general, several approximations called *estimated dependency graphs (EDGs)* have been proposed. NaTT implements the EDG proposed in [8].

2.2 Reduction Pair Processor

This processor forms the core of NaTT. A *reduction pair* is a pair $\langle \succsim, \succ \rangle$ of orders s.t. \succ is *compatible* with \succsim (i.e., $\succsim \cdot \succ \cdot \succsim \subseteq \succ$), both of \succsim and \succ are stable under

substitution, \succsim is monotone and \succ is well-founded. From a DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$, if all the involved rules are weakly decreasing (i.e., $\mathcal{P} \cup \mathcal{R} \subseteq \succsim$), strictly decreasing rules in \mathcal{P} (w.r.t. \succ) can be removed. A great number of techniques for obtaining reduction pairs have been proposed so far. **NaTT** supports the following ones:

- Some *simplification orders* combined with *argument filters* [1]:
 - the *Knuth-Bendix order (KBO)* [15] and its variants including KBO with *status* [20], the *generalized KBO* [19] and the *transfinite KBO* [18, 22],
 - the *recursive path order* [3] and the *lexicographic path order (LPO)* [14],
- *polynomial interpretations (POLO)* [1, 17] and its variants, including certain forms⁵ of *POLO* with negative constants [11] and *max-POLO* [6],
- the *matrix interpretation method* [4, 13], and
- the *weighted path order (WPO)* [25, 26].

Note that all of the above mentioned reduction pairs are subsumed by WPO. That is, by implementing WPO we obtain the other reduction pairs for free. We discuss the implementation details in Section 3.

2.3 Rule Removal Processor

In the worst case, the size of dependency pairs is quadratic in the size of the input TRS \mathcal{R} . Hence it is preferable to reduce the size of \mathcal{R} before computing dependency pairs. To this end **NaTT** applies the *rule removal processor* [7]. If all rules in \mathcal{R} are weakly decreasing w.r.t. a *monotone* reduction pair, then the processor removes strictly decreasing rules from \mathcal{R} . The required monotonicity of a reduction pair is obtained by choosing appropriate parameters for the implementation of WPO described above.

2.4 Uncurrying Processor

Use of uncurrying for proving termination is proposed for *applicative* rewrite systems in [12]. The uncurrying implemented in **NaTT** is similar to the generalized version proposed in [21], in the sense that it does not assume *application symbols* to be binary. A symbol f is considered as an application symbol if all the following conditions hold:

- f is defined and has positive arity,
- a subterm of the form $f(x, \dots)$ does not occur in any left-hand-sides of \mathcal{R} ,
- a subterm of the form $f(g(\dots), \dots)$ occurs in some right-hand-side of \mathcal{R} .

If such application symbols are found, then \mathcal{R} is uncurried w.r.t. the uncurrying TRS \mathcal{U} that consists of the following rules:⁶

$$f(f^l g(x_1, \dots, x_m), y_1, \dots, y_n) \rightarrow f^{l+1} g(x_1, \dots, x_m, y_1, \dots, y_n)$$

for every $g \neq f$ and l less than the *applicative arity*⁷ of g , where $f^0 g$ denotes g and $f^{l+1} g$ is a new function symbol of arity $m + n$.

⁵ Here, negative values are allowed only for the constant part.

⁶ The notation is derived from the *freezing* technique [23].

⁷ Applicative arities are taken so that η -saturation is not needed.

3 The Weighted Path Order

As we mentioned in the introduction, **NaTT** implements only WPO for obtaining reduction pairs. WPO is parameterized by (1) a *weight algebra* which specifies how weights are computed, (2) a *precedence* on function symbols, and (3) a *status function* which specifies how arguments are compared. In the following sections, we present some options which **NaTT** provides for specifying search spaces for these parameters.

3.1 Templates for Weight Algebras

One of the most important tasks in proving termination by WPO is finding an appropriate weight algebra. In order to reduce the task to an SMT problem, **NaTT** considers *template algebras* over integers. Currently the following template algebras are implemented:

- The algebra \mathcal{Pol} indicates that weights of terms are computed by a linear polynomial. Interpretations are in the following shape:

$$f_{\mathcal{Pol}}(x_1, \dots, x_n) = w_f + \sum_{i=1}^n c_{f,i} \cdot x_i \quad (1)$$

where the *template variables* w_f and $c_{f,1}, \dots, c_{f,n}$ should be decided by an external SMT solver.

- The algebra \mathcal{Max} indicates that weights are computed using the max operator. A symbol f with arity ≥ 1 is interpreted in the following shape:

$$f_{\mathcal{Max}}(x_1, \dots, x_n) = \max_{i=1}^n (p_{f,i} + c_{f,i} \cdot x_i) \quad (2)$$

where $p_{f,1}, \dots, p_{f,n}$ are template variables. For constant symbols, interpretations of the shape (1) are used. Since the operator max is not usually supported by SMT solvers, these interpretations are encoded as quantifier-free formulas using the technique presented in [25].

- The algebra \mathcal{MPol} combines both forms of interpretations described above. Since it is inefficient to consider all combinations of these interpretations, \mathcal{MPol} decides the shape of interpretations according to the following intuition: If a constraint such as $f(x) > g(x, x)$ appears, then g is interpreted as $g_{\mathcal{Max}}$, because the imposed constraint $c_{f,1} \geq c_{g,1} \wedge c_{f,1} \geq c_{g,2}$ is easier than $c_{f,1} \geq c_{g,1} + c_{g,2}$, which would be imposed by the interpretation $g_{\mathcal{Pol}}$.

The template variables introduced above are partitioned into two groups: template variables $w_f, p_{f,1}, \dots, p_{f,n}$ are grouped in the *constant part*, and template variables $c_{f,1}, \dots, c_{f,n}$ are in the *coefficient part*. For efficiency, it is important to properly restrict the range of these variables.

Table 1. Parameters for some monotone reduction pairs.

Technique	template	coefficient	constant	precedence	status
Linear POLO	$\mathcal{P}ol$	\mathbb{Z}_+	\mathbb{N}	no	empty
LPO	$\mathcal{M}ax$	$\{1\}$	$\{0\}$	yes	total
KBO ⁸	$\mathcal{P}ol$	$\{1\}$	\mathbb{N}	yes	total
Transfinite KBO ⁸	$\mathcal{P}ol$	\mathbb{Z}_+	\mathbb{N}	yes	total

3.2 Classes of Precedences

NaTT offers “quasi” and “strict” precedences, as well as an option to disable them (i.e., all symbols are considered to have the same precedence). For reduction pairs using precedences, we recommend quasi-precedences which are chosen by default, as the encoding follows the technique of [27] that naturally encodes quasi-precedences.

3.3 Classes of Status Functions

NaTT offers three classes of *status functions*: “total”, “partial” and “empty” ones. The standard notions of *status functions* are total ones that were introduced to admit *permutation* of arguments when comparing them lexicographically from left to right (cf. [20]). Such a comparison appears in many well-known reduction pairs; famous examples are LPO and KBO. By combining the idea of argument filters, status functions have recently been generalized to *partial* ones, that do not only *permute* but may also drop some arguments [24]. A partial status is beneficial for KBO, and even more significant when combined with WPO [26]. The extreme case of a partial status is the “empty” status, that drops all arguments and so no comparison of arguments will be performed. This option corresponds to the nature of interpretation methods, e.g. POLO, if precedences are also disabled.

3.4 Obtaining Well-known Reduction Pairs

Although most of the existing reduction pairs are subsumed by WPO, some of them are still useful for improving efficiency, due to the restricted search space and simplified SMT encoding. We list parameters that correspond to some known reduction pairs in Tables 1 and 2. Note here that the effects of non-collapsing argument filters are simulated by allowing 0-coefficients in the weight algebra. Thus NaTT has a dedicated implementation only for *collapsing* argument filters, and implementations of usable rules for interpretation methods and path orders are smoothly unified.

⁸ Further constraints for *admissibility* are imposed.

Table 2. Parameters for some (non-monotone) reduction pairs.

Technique	template	coefficient	constant	precedence	status
Linear POLO	$\mathcal{P}ol$	\mathbb{N}	\mathbb{N}	no	empty
Max-POLO	$\mathcal{M}Pol$	\mathbb{N}	\mathbb{Z}	no	empty
LPO + argument filter	$\mathcal{M}ax$	$\{0, 1\}$	$\{0\}$	yes	total
KBO + argument filter	$\mathcal{P}ol$	$\{0, 1\}$	\mathbb{N}	yes	total
Matrix interpretations	$\mathcal{P}ol$	$\mathbb{N}^{d \times d}$	\mathbb{N}^d	no	empty
WPO($\mathcal{M}Sum$)	$\mathcal{M}Pol$	$\{0, 1\}$	\mathbb{N}	yes	partial

4 Cooperation with SMT Solvers

NaTT is designed to work with any SMT-LIB 2.0 compliant solvers that support at least **QF_LIA** logic, for which various efficient solvers exist.⁹ **NaTT** extensively uses SMT encoding techniques for finding appropriate reduction pairs; the conditions of reduction pair processors are encoded into the following SMT constraint:

$$\bigwedge_{l \rightarrow r \in \mathcal{R}} \llbracket l \succsim r \rrbracket \wedge \bigwedge_{s \rightarrow t \in \mathcal{P}} \llbracket s \succsim t \rrbracket \wedge \bigvee_{s \rightarrow t \in \mathcal{P}} \llbracket s \succ t \rrbracket \quad (3)$$

where each $\llbracket l \succsim r \rrbracket$ is an SMT formula that represents the condition $l \succsim r$. In the remainder of this section, we present two techniques for handling such constraints that contribute to the efficiency of **NaTT**.

4.1 Use of Interactive Features of SMT Solvers

In a typical run of termination verification, constraints of the form (3) are generated and solved many times, and each encoding sometimes involves thousands of lines of SMT queries with a number of template and auxiliary variables. Hence runtime spent for the SMT solver forms a large part of the overall runtime of the tool execution. **NaTT** tries to reduce the runtime by using *interactive* features of SMT solvers,¹⁰ which are specified in SMT-LIB 2.0.

For each technique of reduction pairs, the encoded formula of the constraint $\bigwedge_{l \rightarrow r \in \mathcal{R}} \llbracket l \succsim r \rrbracket$ need not be changed during a run, as far as \mathcal{R} is not modified.¹¹ Hence, when a reduction pair processor is applied for the first time, the back-end SMT solver is initialized according to the following pseudo-script:

```
(assert ( $\bigwedge_{l \rightarrow r \in \mathcal{R}} (u_{l \rightarrow r} \Rightarrow \llbracket l \succsim r \rrbracket)$ ))
(push)
```

⁹ Cf. the Satisfiability Modulo Theories Competition, <http://smtcomp.org/>.

¹⁰ **NaTT** is not the first tool to use the interactive features of SMT solvers. For example, the Houdini implementation in Boogie uses the features [16].

¹¹ Although rules in \mathcal{R} may be removed by considering *usable rules*, the formula still need not be changed, since it can be simulated by negating a propositional variable that represents whether the rule is usable or not.

where $u_{l \rightarrow r}$ is a boolean variable denoting whether the rule $l \rightarrow r$ is usable or not. When the processor is applied to an SCC \mathcal{P} , the following script is used:

```
(assert ( $\bigwedge_{s \rightarrow t \in \mathcal{P}} \llbracket s \succsim t \rrbracket \wedge \bigvee_{s \rightarrow t \in \mathcal{P}} \llbracket s \succ t \rrbracket$ ))
(check-sat)
```

Then, if a solution is found by the SMT solver, **NaTT** analyzes the solution using the `get-value` command. After this analysis, the command

```
(pop)
```

is issued to clear the constraints due to \mathcal{P} and go back to the context saved by the `(push)` command. In order to derive the best performance of the solver,

```
(reset)
```

is also issued in case sufficiently many rules become unusable (e.g., 1/3 of the rules in \mathcal{R}) from \mathcal{P} . All these commands, `push`, `pop` and `reset` are expected to be available in SMT-LIB 2.0 compliant solvers.

4.2 Use of Linear Arithmetic

Note that expressions of the form (1) or (2) are nonlinear, due to the coefficients $c_{f,1}, \dots, c_{f,n}$. However, not many SMT solvers support *nonlinear* arithmetic, and even if they do, they are much less scalable than they are for linear arithmetic. Hence, we consider reducing the formulas to linear ones by restricting the range of $c_{f,1}, \dots, c_{f,n}$ e.g. to $\{0, 1\}$. Although the idea is inspired by [2], **NaTT** uses a more straightforward reduction using `ite` (*if-then-else*) expressions. Each coefficient $c_{f,i}$ is replaced by the expression `(ite $b_{f,i}$ 1 0)` where $b_{f,i}$ is a propositional variable, and then multiplications are reduced according to the rule:

$$(* (ite\ e_1\ e_2\ e_3)\ e_4) \rightarrow (ite\ e_1\ (*\ e_2\ e_4)\ (*\ e_3\ e_4))$$

It is easy to see that this reduction terminates and linearizes expressions of the form (1) or (2). It is also possible to avoid an explosion of the size of formulas by introducing a auxiliary variable for the duplicated expression e_4 .

Example 1. Consider the constraint $f(f(a)) > b$ interpreted in the algebra \mathcal{Pol} , and suppose that the range of $c_{f,1}$ is restricted to $\{1, 2\}$. The interpretation of the term $f(f(a))$ is reduced as follows (written as S-expressions):

$$\begin{aligned} \llbracket f(f(a)) \rrbracket &= (+\ w_f\ (*\ (ite\ b_{f,1}\ 2\ 1)\ \llbracket f(a) \rrbracket)) \\ &\rightarrow (+\ w_f\ (ite\ b_{f,1}\ (*\ 2\ \llbracket f(a) \rrbracket)\ \llbracket f(a) \rrbracket)) \end{aligned}$$

Similarly, for $f(a)$ we obtain

$$\llbracket f(a) \rrbracket \rightarrow (+\ w_f\ (ite\ b_{f,1}\ (*\ 2\ w_a)\ w_a))$$

Now, the constraint $\llbracket f(f(a)) > b \rrbracket$ is expressed by the following script:

```
(define-fun v (+ wf (ite bf,1 (* 2 wa) wa)))
(assert (> (+ wf (ite bf,1 (* 2 v) v) wb)))
```

In contrast to SAT encoding techniques [4–6], we do not have to care about the bit-width for the constant part and intermediate results. It is also possible to indicate that **NaTT** should keep formulas nonlinear, and solve them using SMT solvers that support **QF_NIA** logic. Our experiments on TPDB¹² problems, however, suggests that use of nonlinear SMT solving is impractical for our purpose.

5 Design

The source code of **NaTT** consists of about 6000 lines of code written in OCaml.¹³ About 23% is consumed by interfacing SMT solvers, where some optimizations for encodings are also implemented. Another 17% is for parsing command-lines and TRS files. The most important part of the source code is the 40% devoted to the implementation of WPO, the unified reduction pair processor. Each of the other processors implemented consumes less than 3%. For computing SCCs, the third-party library **ocamlgraph**¹⁴ is used.

5.1 Command Line Interface

The command line of **NaTT** has the following syntax:

```
./NaTT [FILE] [OPTION] ... [PROCESSOR] ...
```

To execute **NaTT**, an SMT-LIB 2.0 compliant solver must be installed. By default, **z3** version 4.0 or later¹⁵ is supposed to be installed in the path. Users can specify other solvers by the `--smt "COMMAND"` option, where the solver invoked by **COMMAND** should process SMT-LIB 2.0 scripts given on the standard input.

The TRS whose termination should be verified is read from either the specified **FILE** or the standard input.¹⁶ Each **PROCESSOR** is either an order (e.g. **POLO**, **KBO**, **WPO**, *etc.*, possibly followed by options), or a name of other processors (**UNCURRY**, **EDG**, or **LOOP**). Orders preceding the **EDG** processor should be monotone reduction pairs and applied as rule removal processors before computing the dependency pairs. Orders following the **EDG** processor are applied as reduction pair processors to each SCC in the EDG. A list of available **OPTIONS** and **PROCESSORS** can be obtained via **NaTT --help**.

¹² The Termination Problem Data Base, <http://termination-portal.org/wiki/TPDB>.

¹³ <http://caml.inria.fr/>

¹⁴ <http://ocamlgraph.lri.fr/>

¹⁵ <http://z3.codeplex.com/>

¹⁶ The format is found at <https://www.lri.fr/~marche/tpdb/format.html>.

Table 3. Effects of the optimizations.

option	yes	no	maybe	T.O.	time
non-linear	767	170	368	158	13138.11
linearized	848	173	429	13	2161.43
interactive	848	173	429	13	1865.50

5.2 The Default Strategy

In case no **PROCESSOR** is specified, the following default strategy will be applied:

- As a rule removal processor, POLO with coefficients in $\{1, 2\}$ and constants in \mathbb{N} is applied.
- Then the uncurrying processor is applied.
- The following reduction pair processors are applied (in this order):
 1. POLO with coefficients in $\{0, 1\}$ and constants in \mathbb{N} ,
 2. algebra *Max* with coefficients in $\{0, 1\}$ and constants in \mathbb{N} ,
 3. LPO with quasi-precedence, status and argument filter,
 4. algebra *MPol* with coefficients in $\{0, 1\}$ and constants in \mathbb{Z} ,
 5. WPO with quasi-precedence, partial status, algebra *MPol*, coefficients in $\{0, 1\}$ and constants in \mathbb{N} ,
 6. matrix interpretations with $\{0, 1\}^{2 \times 2}$ matrices and \mathbb{N}^2 vectors.
- If all the above processors fail, then a (naive) loop detection is performed.

6 Assessment

In this section, we verify the significance of the contributions of **NaTT** by experiments and by its result in the termination competition.

6.1 Effects of Optimizations

First, we verify the effect of the optimizations proposed in Section 4. The experiments are run on a server equipped with a quad-core Intel Xeon E5-3407v2 processor running at a clock rate of 2.40GHz and 32GB of main memory. As the SMT solver, we choose **z3** 4.3.2.

In Table 3, we compare the following options of **NaTT**.

- The ‘non-linear’ row considers interpretations of the non-linear shape of (1) and (2), and directly solves the encoded problem via **QF-NIA** logic. To achieve a practical runtime, the constant part is bounded by upper bound 3.
- The ‘linearized’ row applies the linearization proposed in Section 4.2.
- The ‘interactive’ row further uses the interactive features as proposed in Section 4.1. This option is the default of **NaTT**.

In the table, we observe a dramatic improvement by the linearization of Section 4.2. The use of interactive features of SMT solvers may look less significant, but the runtime improves by almost 10%.

6.2 Results in the Termination Competition

Many tools have been developed for proving termination of TRSs, and the international termination competition has been held annually for a decade. **NaTT** participated in the *TRS Standard* category of the full-run 2013,¹⁷ where the other participants are versions of: **AProVE**,¹⁸ **T_TT₂**,¹⁹ **MU-TERM**,²⁰ and **WANDA**.²¹ Using the default strategy described in Section 5.2, **NaTT** (dis)proves termination of 982 TRSs (unfortunately, the competition version of **NaTT** failed to input 36 problems due to a bug in parser) and comes next to (the two versions of) **AProVE**, the constant champion of the category. It should be noticed that **NaTT** proved termination of 34 TRSs out of the 159 whose termination could not be proved by any other tool. **NaTT** is notably faster than the other competitors; it consumed only 21% of the time compared to **AProVE**, the second fastest. We expect that we can further improve efficiency by optimizing to multi-core architecture; currently, **NaTT** runs in almost single thread.

NaTT also participated in the *SRS Standard* category. However, the result is not as good as it is for TRSs. This is due to the fact that the default strategy of Section 5.2 is designed only for non-unary signatures. Indeed, when a unary symbol is considered, an interpretation of the form (2) is equivalent to one of the form (1). It should be improved by choosing a strategy depending on the shape of input TRSs.

7 Conclusion

We described the implementation and techniques of the termination tool **NaTT**. The novel implementation of the weighted path order is described in detail, and some techniques for cooperating SMT solvers are presented. Together with these efforts, **NaTT** is one of the most efficient and strongest tools for proving termination of TRSs.

Because of its efficiency, **NaTT** is especially strong on larger systems. In general, a larger input TRS requires a larger proof script to be produced, which is quite difficult to be checked by hand. Thus our future work is to produce proofs in the *certifiable proof format*.²²

Acknowledgments We thank the anonymous reviewers of this paper for careful inspections and constructive comments that improved the quality of this paper. We also thank Shaz Qadeer for information about Boogie. This work was supported by JSPS KAKENHI #24500012.

¹⁷ <http://termcomp.uibk.ac.at/>

¹⁸ <http://aprove.informatik.rwth-aachen.de/>

¹⁹ <http://cl-informatik.uibk.ac.at/software/ttt2/>

²⁰ <http://zenon.dsic.upv.es/muterm/>

²¹ <http://wandahot.sourceforge.net/>

²² <http://cl-informatik.uibk.ac.at/software/cpf/>

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000.
2. C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In *Proc. CADE '09*, LNCS 5663, pages 294–305, 2009.
3. N. Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17(3):279–301, 1982.
4. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reasoning*, 40(2-3):195–220, 2008.
5. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT '07*, LNCS 4501, pages 340–354, 2007.
6. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal termination. In *Proc. RTA '08*, LNCS 5117, pages 110–125, 2008.
7. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*, LNAI 3452, pages 75–90, 2004.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. FroCoS '05*, LNAI 3717, pages 216–231, 2005.
9. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *J. Autom. Reasoning*, 37(3):155–203, 2006.
10. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proc. RTA '04*, LNCS 3091, pages 249–268, 2004.
11. N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proc. AISC '04*, LNAI 3249, pages 185–198, 2004.
12. N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for termination and complexity. *J. Autom. Reasoning*, 50(3):279–315, 2013.
13. D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. RTA '06*, LNCS 4098, pages 328–342, 2006.
14. S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering, 1980. Unpublished note.
15. D.E. Knuth and P. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, New York, 1970.
16. A. Lal, S. Qadeer, and S. Lahiri. Corral: A solver for reachability modulo theories. In *Proc. CAV '12*, LNCS 7358, pages 427–443, 2012.
17. D. Lankford. On proving term rewrite systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, 1979.
18. M. Ludwig and U. Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In *Proc. LPAR '07*, LNAI 4790, pages 348–362, 2007.
19. A. Middeldorp and H. Zantema. Simple termination of rewrite systems. *Theor. Comput. Sci.*, 175(1):127–158, 1997.
20. J. Steinbach. Extensions and comparison of simplification orders. In *Proc. RTA '89*, LNCS 355, pages 434–448, 1989.
21. C. Sternagel and R. Thiemann. Generalized and formalized uncurrying. In *Proc. FroCoS '11*, LNAI 6989, pages 243–258, 2011.

- 22. S. Winkler, H. Zankl, and A. Middeldorp. Ordinals and Knuth-Bendix orders. In *Proc. LPAR '12*, LNCS ARCoSS 7180, pages 420–434, 2012.
- 23. H. Xi. Towards automated termination proofs through “freezing”. In *Proc. RTA '98*, LNCS 1379, pages 271–285, 1998.
- 24. A. Yamada, K. Kusakari, and T. Sakabe. Partial status for KBO. In *Proc. WST '13*, pages 74–78, 2013.
- 25. A. Yamada, K. Kusakari, and T. Sakabe. Unifying the Knuth-Bendix, recursive path and polynomial orders. In *Proc. PPDP '13*, pages 181–192, 2013.
- 26. A. Yamada, K. Kusakari, and T. Sakabe. A unified order for termination proving. *CoRR*, abs/1404.6245, 2014. Submitted to SCP.
- 27. H. Zankl, N. Hirokawa, and A. Middeldorp. KBO orientability. *J. Autom. Reasoning*, 43(2):173–201, 2009.